4. a) O echipa de programatori a dezvoltat algoritmul de lock prezentat in pseudocodul urmator. *ThreadId* se considera a fi o clasa ce furnizeaza un id unic pozitiv fiecarui thread.
Intr-o executie concurenta a n > 1 thread-uri, este acest algoritm starvation-free? Argumentati.

```
1 class ShadyLock {
2    private volatile int turn;
3    private volatile boolean used = false;
4
5    public void lock() {
6       int me = ThreadId.get();
7       do {
8          do {
9                turn = me;
10         } while (used);
11         used = true;
12      } while (turn != me);
13   }
14
15   public void unlock () {
16      used = false;
17   }
18 }
```

Un algoritm cu proprietatea starvation-free garanteaza ca toate threadurile ce incearca sa acceseze sectiunea critica vor obtine acces intr-un timp finit si vor progresa, fara a risca sa ramana blocate sau sa astepte la nesfarsit din cauza altor threaduri concurente.

In clasa ShadyLock, accesul la linia *turn = me* se realizeaza fara nicio verificare prealabila sau mecanism de prioritizare intre threaduri. Acest lucru poate duce la rescrierea frecventa a variabilei *turn* de catre diferite threaduri, provocand o situatie in care un fir poate fi permanent blocat in bucla *while (turn != me);* . Acest blocaj apare deoarece threadul respectiv nu reuseste sa pastreze valoarea turn pana la accesul efectiv in sectiunea critica, ceea ce face ca algoritmul sa nu fie starvation-free.

Trace demonstrativ:

1. Un fir de executie T1 seteaza *turn = 1* si intra in bucla *while (used);*
2. Un alt fir de executie, T2, seteaza *turn = 2* si intra in bucla *while (used);* inainte ca T1 sa fi setat used = true;
3. Daca used este false si T2 seteaza *used = true* mai rapid decat T1, T2 va reusi sa intre in sectiunea critica, dar T1 va ramane blocat in bucla *while (turn != me)*.

In acest mod, T1 poate ramane in stare de starvation, deoarece nu exista niciun mecanism care sa-i garanteze accesul la sectiunea critica intr-un timp finit.